Issues in Platform-Independent Support for Multimedia Desktop Conferencing and Application Sharing

O. Kim, P. Kabore, J.P. Favreau and H. Abdel-Wahab[*]
Multimedia & Digital Video Technologies Group
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, Md 20899
(okim,kabore,favreau,wahab)@snad.ncsl.nist.gov

Abstract

Although Multimedia desktop conferencing and application sharing among geographically dispersed users are increasingly popular modalities, their spread is inhibited by platform-dependency problems. In this paper, an approach which exploits the use of the Java programming language to accommodate different hardware and window systems is investigated and a prototype is implemented. Our approach is based on *replicated tool architecture* in which each participant runs a copy of the application and the activity of each user is multicast to all the participants in the conference. The problems associated with this approach such as view synchronization and replicated object management are among the issues addressed in our research. In addition, we are developing standard functions and mechanisms that allow conference participants to seamlessly use the audio and video features available on most PC's and workstations. Our research on multimedia stream synchronization and adaptation, the incorporation of reliable multicasting and the development of distributed control algorithms are expected to result in increased conference quality, performance and robustness.

KEY WORDS: High Bandwidth Applications, Multimedia Communications, Java, Interoperability, Computer Supported Cooperative Work, Desktop Conferencing, Multicasting, Distributed Systems.

[*]NIST faculty and Professor of Computer Science at Old Dominion University, Norfolk, Virginia

1 Introduction

With the proliferation of high bandwidth computer networks and powerful multimedia workstations, it is now feasible to build collaborative systems that allow users to have real-time interaction with each other and remotely work together as a team. In addition to using audio and video we believe that it is very productive if all participants simultaneously have full access to their shared computer-stored materials and have the ability to share and manipulate them together.

Most current existing collaborative systems require the participants in a conference to use the same window system. For example, XTV [2, 3] and Suite [9] are based on the X window system and require that the participant's machines run the X server. Other systems such as WTV [6] have tried to replicate the functionality of XTV replacing the X windows with Microsoft Windows. Ideally, each participant in a collaborative conference should be able to use whatever platform he or she prefers. For example, some may use PCs running MS Windows 95. Others may use workstations running different version of UNIX and X windows, yet others may use PowerPC Macintoshs. Before the introduction of Java, this sort of collaboration was enormously difficult to achieve. Java programs are compiled to an architecture neutral byte-code format and thus can run on any system that implements a Java virtual machine and its abstract window system. Java provides a fortuitous opportunity for the Computer Supported Cooperative Work (CSCW) [12] community to overcome a barrier which hitherto hindered the wide spread use of collaboration technology.

To overcome the platform-dependency problem for application sharing in heterogeneous platforms, NIST (National Institute of Standards and Technology) and ODU (Old Dominion University) are jointly conducting a research project to investigate mechanisms for sharing multimedia applications among participants on not only heterogeneous windowing and operating systems, but on different hardware platforms.

We have developed mechanisms to intercept, distribute and recreate the user events that allow single-user Java applications to be shared, without modifications, among conference participants. These mechanisms can be run transparently on any system implementing Java. The mechanisms incorporate the services of network communications, conference management and floor control management. The network communications services include distribution of the data among conference participants; conference management includes joining and leaving a session; and floor control includes participant's control and interaction with the application during a session.

In this paper, we refer to the prototype which has been developed as the Java Collaborative Environment (JCE). We are now in the process of augmenting JCE to include both audio and

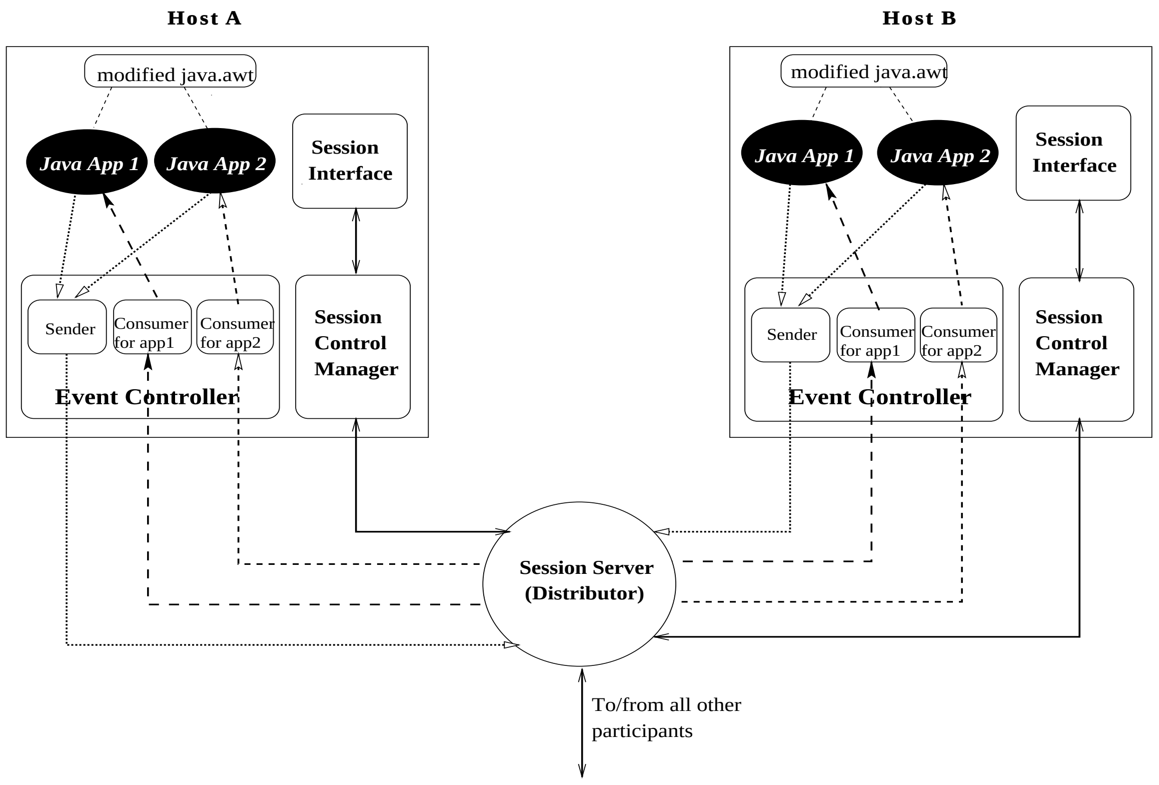

Figure 1: **Overall System Architecture**

video in an integrated platform-independent desktop conferencing system.

The remainder of the paper is organized as follows: Section 2 describes the JCE system architecture and provides an overview of its major components and functions. Section 3 discusses the problems associated with executing multiple replicated copies of the same programs. In Section 4 we show some applications for event logging: late-joining and playback. Section 5 concentrates on communications issues such as the use of reliable multicasting, conference information service and conference control. Support for platform-independent audio and video is discussed in Section 6. The quality of session attributes such as multi-media synchronization and adaptation are presented in Section 7. Finally, Section 8 gives our conclusions and future work.

2 JCE System Architecture

The JCE is a framework for shared interactive multimedia applications. Figure 1 depicts the overall system architecture, and the relationship and communication paths among all processes of the system, for a given conferencing session.

The Java applications denoted as *Java App 1 (and 2)* in Figure 1 are not part of the system. They are collaboration-unaware single-user applications developed using the standard java.awt package [18]. Participants can invoke one or more applications in a given conference. Our model is based on the *replicated architecture* [13] in which an instance of each application runs locally at each participant's site and the activity of each user is distributed to all the participants in a conference.

As shown in Figure 1 the JCE provides a *modified java.awt* class library and consists of three components: the *Session Control Manager* and its Interface, the *Event Controller*, and the *Session Server*. These components are discussed in the next two subsections.

2.1 Modified Java Windowing Package

In the standard java.awt package [18], **Component** class is the superclass of all the GUI components and it contains the user event handling. The unmodified *handleEvent* method in **Component** class processes the user events. We have modified this *handleEvent* method to provide a mechanism that intercepts the user events from each application, and sends them to the Session Server, giving all participants the same application state.

The following code fragments show the modified *handleEvent* method in **Component** class, with modifications shown in *italic*.

```
import EventController;

public class Component implements ImageObserver {
    // existing code
    public boolean handleEvent(Event evt) {

    if (!EventController.sender(evt))
        return false;

    switch(evt.id) {
      // remaining existing code
    }
    return false;
  }
}
```

This approach allows Java single-user applications to be shared, and enables their simultaneous viewing and collaboration among conference participants.

2.2 System Components

The *Session Control Manager* (SCM) and the *Session Interface* combined provide the user with a graphical interface offering the following options: to call, join or leave a session; to start applications; and to request or release a floor. Each participant is given an SCM process that exchanges control information with the Server for the duration of the session.

The *Event Controller* is the core of the collaboration mechanisms. It is composed of two processes: the event *Sender* and *Consumer*. When an application is started an *Event Controller* for the application is automatically instantiated by the Session Control Manager. When two or more applications are shared, two or more *Consumers* are created as shown in Figure 1. The *Sender* is declared as a static (i.e., class) method of the *Event controller*, so only one *Sender* method exists for all applications. The *Sender* method first checks the intercepted event to determine whether or not it should be sent to the *Session Server*, since events originating from shared applications are always forwarded. The *Consumer* processes receive events redistributed by the Session Server from other participants, and post them to the local instance of the application as if they were originated locally. This process is completely transparent to the application, i.e., the application is unaware that it is being shared.

The *Session Server* in Figure 1 provides three distinct functions: distribution of all messages to all participants; group management for a given session, including joining or leaving a session; and server floor control management.

2.3 Alternative Implementations

Besides the modifications to the standard java.awt package [18] which allow existing single-user applications to be shared, as detailed above, extensions to the standard package have been developed which allow programmers to develop new collaboration applications or modify existing single-user applications [4]. The advantages of each approach are noted below.

Advantages of *Modified* Library

The existing and new single-user applications can be shared transparently, so that application developers do not have to be concerned about whether the applications are collaboration-aware.

Further, since **Component** class is the superclass of all the GUI components, the JCE need not be updated when new GUI components are developed and introduced. In contrast, the *extended* libraries must be updated to account for the new components.

Advantages of *Extended* Library

This method provides more efficiency and flexibility in object event handling in shared applications. Each GUI component derived from the **Component** superclass handles its own user events, thus eliminating those events coming from other than shared applications such as the *Session Control Manager.*

Moreover, the *extended* library requires that no change needs to be made to the environment by changing the **CLASSPATH** variable, whereas the use of the *modified* library requires that the new java.awt package must be installed and used at runtime.

3 Replication Management

Most applications need to create or use objects during execution, for instance, the environment variable, the initialization dot files, and the files storing multimedia data. These objects must be replicated and available at each site for the correct operation of the JCE system. There are three types of objects to be replicated and managed: environment, operational and final objects.

3.1 Environment Objects

In order to enforce strict WYSIWIS (What You See Is What I See), each replica of the shared application must have the same *operating environment* (e.g., in UNIX/X systems terminology, each site should have the same environment variables, same initialization files and the same X resource files). Before the invocation of each copy of the shared application at each site, we must ensure that all sites have identical operating environments. Since each application may have a specific and different operating environment from other applications, any solution to this problem requires obtaining specific information about each application. This can be achieved by having an *environment profile* for each shared application that contains the operating environment specification and default locations where the values of these resources can be obtained, replicated and installed at each participant's site prior to the execution of the application. It is prudent to save and later restore the original operating environment so that the user can run the application in single-user mode according to his/her own preferred settings of the application. This concept of common operating environment and profile for each shared application is important to guarantee full and strict WYSIWIS behavior. This does not preclude the participants from sharing applications in which users see the same data in different ways such as local selection of font styles and sizes according to each user's preference.

3.2 Operational Objects

The second set of objects needed during the life-time of shared applications is the *operational files*. These files may include data, images, audio clips, video clips, etc. that are needed during the execution of the program. If these files are known and available in advance, then we can specify an *operational profile* for each application that contains a list of these files and a default location where it can be obtained. Prior to the execution of the application, these files are distributed and installed in the appropriate "well-known" directories. Again, one ought not to destroy or alter the original copies of these files so they can be restored upon the termination of the shared application.

3.3 Final Objects

The third set of objects is the newly created files or the files to be modified during the shared application life-span. In this case a *final profile* is used to list these files and specify whether each participant should keep a copy. It may be necessary (e.g., for integrity or security reasons) to specify for some files that only one site should keep a copy and that all other copies of these files should be deleted.

4 Late Comers, Recording and Playback

Saving all input events to each shared application by the conference server is called *event logging* and this is similar to transaction logging in database systems [10]. There are many applications for event logging in JCE:

1. *Late Comers*: Participants who join an ongoing conference after the start of at least one shared application are called *Late-Comers* [8]. Although in JCE, participants may join the conference any time after it starts, they may not have the same view for those shared applications which started before they joined. To bring the late-comers views in synchronization with all other participants, we may send all the logged events to their instances of those shared applications. The time it takes to achieve this synchronization of views is a function of the number of the accumulated events. This in turn depends on how late the participants join and how much activity occurred before they join. An open research issue is to investigate whether it is necessary to send all the logged events or only send a small fraction of these events similar to what XTV does in its handling of late-comers [8].

2. *Recording/Playback*: Event logging is considered to be a form of session recording. To playback a recorded session, all it takes is to start an instance of each involved application and feed it with the events saved in the log file. The playback may be seen by a single person or by all the participants in a conference like any other "live" shared application. This can be very useful in many applications such as:

   - to investigate why an application has crashed and what is the sequence of events that have led to it.
   - to use it as a teaching aid by recording the steps of interaction with an application which users may view at a later time.

5 Communication and Distributed Control Issues

This section is devoted to issues of system performance, usability and robustness. To increase the system performance as the number of participants increases we should use reliable multicasting for data transport instead of the current server-based (star-topology) TCP connections [16]. To make it easy for participants to use the system and join any on-going conferences or start new conferences, we should use the services provided by CIS, a real-time, internet-based Conferencing Information Server, as discussed in Section 5.2. A robust conferencing system should not depend on one central process for its control so a set of distributed algorithms must be developed to replace the functions performed by the current conference server.

5.1 Use of Reliable Multicasting

In our current implementation, all the participants are connected to the server with TCP connections as shown in Figure 2.

If one participant needs to send a message to all other participants, he or she sends it to the server which in turn distributes it to all participants, one at a time, using the TCP connections. This may be acceptable if the number of participants is small (e.g., 4 or 5), but as the number of participants increases, the system performance degrades and the session quality is reduced, as measured by several parameters, such as view synchronization, to be discussed in Section 7.1.

The use of *reliable multicasting* (e.g., RMP provided by Berkeley/West Virginia [15]) greatly improves both the performance and the quality of session. To use multicasting, each participant and the server will have a UDP socket in addition to the existing TCP sockets as shown in Figure 2. TCP connections are used for one to one communications among the participants and the server. In the current JCE implementation if the server is down the whole

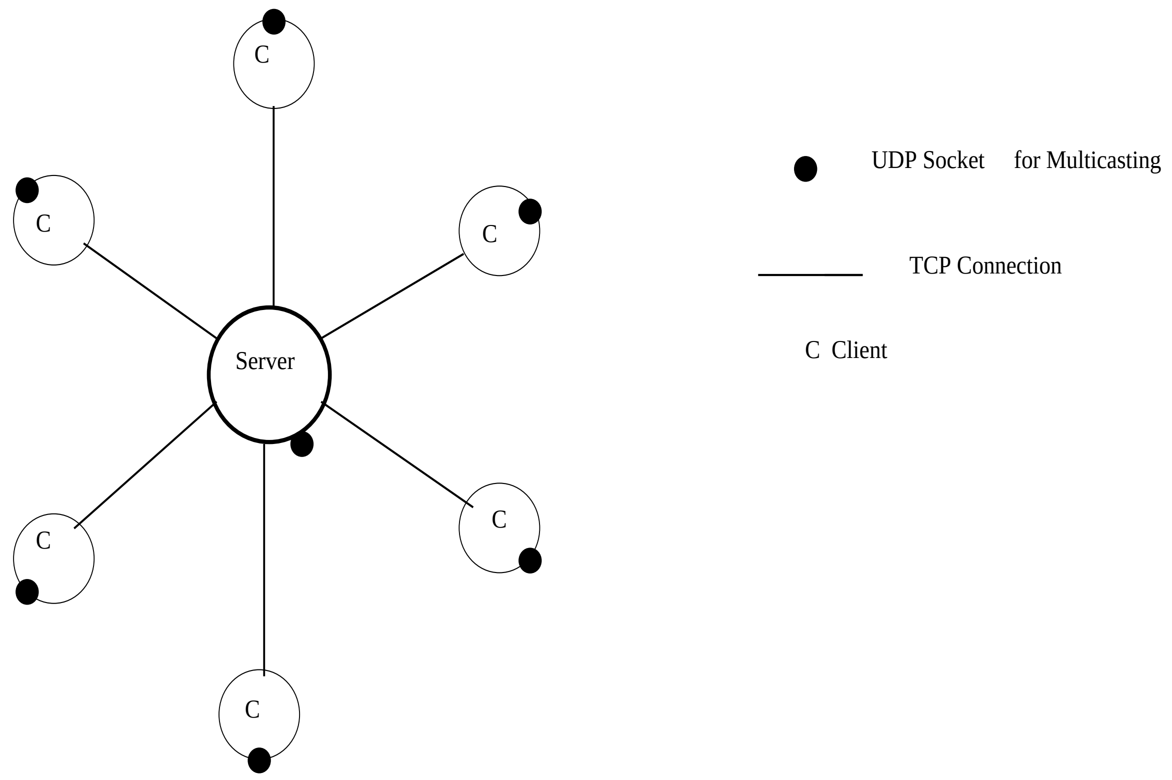

Figure 2: **Communications among Clients and Server**

conference will be terminated and can not continue. However, as discussed in Section 5.3, we plan to distribute the server functions among all the participants and eliminate the need for TCP connections. At that time we will have a truly distributed conferencing system that is not subject to a single point of failure.

5.2 The Conferencing Information Service

In order to facilitate the process of joining a conference, a conferencing information service (CIS) [1] is utilized. This service allows a conference to advertise specific information about itself to help potential participants to find out information about the conference and allow them to join. In order to use CIS, a conferencing system like JCE needs to implement the CIS advertisement protocol and provide an interface that allows users to browse through the information about various conferences and join any selected conference. This interface may be implemented as a stand-alone application or as a Java applet that can be used within an Internet browser.

5.3 Distributed Management of Conference Resources

In our current architecture the server plays a central and vital role in connecting the participants, using the star-topology TCP connections shown in Figure 2, and performing various conference management functions such as floor management. As we have seen earlier, we can use reliable multicasting to replace the role of the server to distribute messages among the participants. However, replacing the conference management function and distributing it among the participants is much harder and requires the design and implementation of a wide variety of distributed algorithms. In particular two algorithms are needed to:

1. maintain an up-to-date list of conference participants and to announce this list to the Conferencing Information Server, and
2. to grant the floor to at most one participant at a time.

6 Platform-Independent Audio and Video Support

Beside shared applications, audio followed by video in this order are important to support full and effective collaboration among participants. In our project, we provide audio support as a standard feature, since the Internet bandwidth may now reasonably support the transport of audio conversations.

Almost all PCs and Workstations now have audio devices (microphone and speakers), though they are often not compatible with each other and may use different audio formats. Thus, our task here is to ensure that all participants can talk and hear each other without worrying about the heterogeneity of their respective audio devices. Our approach to resolving this issue is to identify the most common audio format and configuration of audio devices and save it as a *Common Audio Format and Configuration* (CAFC) file. Whenever a participant joins a session, his/her audio devices are examined to see if they can be configured to the specifications stored in the CAFC file. If it is determined that a participant's device cannot be configured or does not support the common audio format specified in the CAFC file, then an appropriate action such as format translation or quality of service degradation for that participant is taken.

Despite advances in compression technology, video communication requires high bandwidth and not many PCs or workstations are equipped with video cards and cameras, which are still expensive components relative to the basic price of the host machines and must be purchased and installed separately. However, we expect in the near future that Internet bandwidth will increase and the video hardware cost will decrease to the point where desktop video communication will become as common as audio. In a two-party system, it is customary to display the other person's video image. When there are multiple participants, however, determining which participant's video image is to be displayed at which time is a matter to be decided by each specific application. For example, in the Interactive Remote Instruction (IRI) system [5] used for distance learning, the teacher's image is always displayed on each student's workstation and only those students engaged in active discussion with the teacher are displayed, in smaller windows. In a general desktop conferencing system, we would like to provide general mechanisms and protocols that users can configure according to their particular needs and preferences. For example, if someone speaks, his image may be displayed by clicking a button if so desired. In a formal meeting where there is a chairman, the group may decide that the chairman's image be always displayed. This issue of determining how many images to display, the quality and size of each image, and when these images are to be displayed is one of the goals of our project. Our other major goal is to support interoperation among many different and diverse video devices using different hardware platforms. If some participants have no, or incompatible video capabilities, they can still participate in the conference using only the audio channels.

To achieve maximum efficiency, in our implementation, we intend to use the traditional IP multicasting to send audio data among the participants. In addition, the standard IGMP (Internet Group Management Protocol) [14, 17] will be used to manage the process of joining and leaving a conference.

7 Synchronization and Adaptation Issues

In order to improve the conference quality as perceived by the participants we must address the following issues and search for innovative solutions.

7.1 View Synchronization

As we described earlier, the JCE is based on the replicated model [13]; that is - for n participants there are n copies of the same application running concurrently, possibly on different hardware (e.g., from powerful workstations to low-end PCs) and software (e.g., operating systems and window systems). Some of the n participants may be connected by a high-speed Intranet, while others are connected to the global Internet with relatively slow links. In this networking environment, it is inevitable that there will be a skew or lack of synchronization among what all the participants see in the shared application windows. Our objective is to reduce these synchronization problems to a minimum and bring it to an acceptable human tolerance level to preserve the concept of WYSIWIS. This problem does not exist in a two-party conferencing system. However, when a large number of participants is involved, the problem is significant and requires an innovative solution. One such solution is to sense and measure the state of each replica of the shared application. The gathered feedback data can then be used to slow down the flow of events to the faster participants or to speed up the delivery of events to the slower sites.

7.2 Multi-Stream Synchronization

An important problem in multimedia applications (e.g., remote learning, video conferencing and information-on-demand) is the temporal synchronization of continuous and discrete media that have the same or different sources. Streams can be captured at the transmitter and a temporal relation between them established. The playback times, at the destination, for the corresponding streams may differ due to communication and network delays, or the difference between two consecutive schedule times of a process, for example. Temporal synchronization requires the preservation of the temporal dependencies among various media at the destination. For example, consider the following scenario. In a collaborative session, at time t_0, a participant speaks, then, at a later time t_1, he/she starts a shared Java application (e.g., whiteboard). On other participant's workstations, it is not sufficient just to play the streams, temporal synchronization must also be maintained. Within $[t_0, t_1]$, the speaker's audio and video need to be synchronized, and the time independent stream generated by the Java application should be synchronized with the other two continuous streams.

There are two particular issues that need to be addressed for temporal synchronization: *intrastream* synchronization and *interstream* synchronization [13]. Intrastream synchronization policies eliminate jitter when playing a periodic stream. Interstream synchronization policies support orchestrated multimedia presentations, preserving the time dependencies between streams when captured. The relations that specify the temporal dependencies between streams are called *synchronization specification* [13]. In live synchronization, the application at the transmitter is responsible for providing the synchronization specification, while the application at the destination is responsible for providing a synchronized presentation according to this information.

The objective of our work is to provide a flexible and robust solution for the temporal interstream synchronization of time dependent (audio, video) and time independent (text, graphics, shared windows) streams in a multimedia application.

7.3 Inter-Stream Adaptation

In collaborative multimedia systems, there is a need for overall control, beyond the level of quality of service (QoS) of individual streams. The quality of the conference as perceived by the end user, must be determined by the end application. At every instant in time, the quality of the conference depends on the priorities of the on-going streams, from the user's perspective, as well as on the actual QoS offered by the system to each of these streams. The main objective is to keep a collaborative session going, with acceptable overall quality. This is achieved by employing a monitoring mechanism at the application level for monitoring the perceived QoS of each stream. For example, a two way audio-video application may choose to degrade the quality of video only, while keeping the audio quality at the same high level.

A system which is not aware of this inter-stream correlation, may degrade the performance of all streams with an equal proportion, in an attempt to react to the overload situation in a fair way. Moreover, the same application may have different priorities for different streams, at every instant in time. Building on the same example mentioned above, if there were a conversation between two physicians, and at a certain point in the conference, the video image of one of the participants was replaced by a VCR tape playback of an operation, then the application may prefer a degradation in the quality of the audio rather than that of the video, in reaction to any overload situations. In such complex collaborative applications, a compromise in the quality of one stream, in favor of another may not only be due to temporary overload situations, but it may be due to inherent capacity constraints in the system. For instance, a video conferencing application supporting several simultaneous participants, may not find enough network bandwidth, or system processing capability, to send a full motion video stream of each participant, at 30 frames per second. As an alternative, each participant may receive a full motion video stream for the speaker, and lower frame rate video streams for other participants. The previous examples suggest that, in collaborative multimedia systems, there is a need for overall control, beyond the level of QoS of individual streams, for a particular application.

8 Conclusions and Future Work

In this paper we have described our current ongoing research and the major issues and problems associated with developing platform-independent desktop conferencing systems that integrate application sharing, audio, video and conference management functions. Among those issues addressed in some details are replication management, accommodating late comers, session recording and playback, scalability through the use of reliable multicasting in both reliable and unreliable (raw IP) forms, global internet conference information service, the integration of audio and video, and the synchronization and adaptation of multimedia streams.

In addition, we have demonstrated the important role of Java by implementing the Java Collaborative Environment (JCE) prototype for application sharing among diverse systems such as UNIX workstation-based and PC Windows-based systems. The merits of the two alternative collaborative mechanisms developed in JCE, the *modified* and the *extended* approaches are also discussed. Our next goal is to use JCE from Internet browsers such as the Netscape Navigator and the Microsoft Internet Explorer. Due to some of the limitation imposed by Internet browsers and Java Applets for security and other reasons, the participants may not be able to perform certain functions. However, we would like to maximize what can be done through the World-Wide Web, identify those functions that cannot be done through it and help the users to perform these functions via a parallel stand-alone interface.

References

[1] H. Abdel-Wahab, I. Stoica and F. Sultan, "The Design and Implementation of an Internet Conference Information System", To appear in *Journal of Internetworking Research & Experience*, 1996.

[2] H. Abdel-Wahab and M. Feit, "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration", *Proceedings, IEEE TriComm '91: Communications for Distributed Applications & Systems*, Chapel Hill, North Carolina, pp. 159-167, April 1991.

[3] H. Abdel-Wahab and K. Jeffay, "Issues, Problems and Solutions in Sharing X Clients on Multiple Displays", *Journal of Internetworking Research & Experience.* pp. 1-15, Vol. 5, No. 1, March 1994.

[4] H. Abdel-Wahab, B. Kvande, S. Nanjangud, O. Kim and J.P. Favreau, " Using Java for Multimedia Collaborative Applications", To appear in the *PROMS'96: Third International Workshop On Protocols for Multimedia Systems*, 1996.

[5] H. Abdel-Wahab, K. Maly, A. Youssef, E. Stoica, C.M. Overstreet, C. Wild, and A. Gupta, "The Software Architecture and Interprocess Communications of IRI: an Internet-based Interactive Distance Learning System", *WETICE'96*, Stanford, California, June 1996.

[6] D. Adams, "WTV: An MS Windows based Collaborative System", Master's Project Report, Department of Computer Science, Old Dominion University, Dec. 1995.

[7] G. Comell and C. S. Horstmann, *Core Java*, Prentice-Hall, 1996.

[8] G. Chung, K. Jeffay and H. Abdel-Wahab, "Accommodating Latecomers in Shared Window Systems", *IEEE Computers*, pp. 72-74, Vol. 26, No. 1, January 1993.

[9] P. Dewan and R. Chouldhary, "A high-level and flexible framework for implementing multiuser interfaces", *ACM Transaction on Information Systems*, Vol. 10, No. 4, 345-380, (October 1993).

[10] R. Elmasry and Navathe, "Fundamentals of Database Systems," 2nd ed., Addison-Wesley, 1994.

[11] D. Flanagan, *Java in a Nutshell*, O'Reilly & Associates, 1996.

[12] J. Grudin, "Computer-Supported Cooperative Work: History and Focus", *IEEE Computer*, Vol. 27, No. 5, 19-26, (May 1994).

[13] R. Steinmetz and K. Nahrstedt, *Multimedia: Computing, Communications & Applications* Prentice-Hall, 1995.

[14] W. R. Stevens, *TCP/IP Illustrated*, Volume 1, Addison-Wesley, 1994.

[15] B. Whetten, T. Montgomery, and S. Kaplan, "A High Performance Totally Ordered Multicast Protocol", *Theory and Practice in Distributed Systems*, Springer Verlag LCNS 938, 1994.

[16] J. Postel, "Transmission Control Protocol", *IETF RFC 793*, 1981.

[17] S. Deering, "Host Extensions for IP Multicasting", *IETF RFC 1112*, 1989.

[18] *Abstract Windowing Toolkit (AWT) package, Java Developers Kit (JDK) Version 1.0 API*, Sun Microsystems Inc. Mountain View, CA 94043